Multiplexing Low and High QoS workloads in Virtual Environments

Sam Verboven, Kurt Vanmechelen
and Jan Broeckhove

University of Antwerp
Research Group Computational
Modeling and Programming

CoMP

Outline

Outline

Infrastructure management

IT infrastructure management is increasingly relying on virtualization

- Physical machines migrate to Virtual machines
- Deployed hardware agnostically using VMM
- VMM offer flexibility in :
  - Partitioning hardware resources
  - Isolation
  - Suspension
  - Migration
  - ...

Utilization

Virtualized servers require guaranteed availability and performance (high-QoS requirements)

- Static resource allocation
- Provisioning resources based on worst case requirements
- Resource usage varies
- Underutilized infrastructure

Utilization

How can we address underutilization?
Dynamically add low priority, low-QoS workloads

- Fill underutilized periods
- Virtualization gives flexibility
  - Start, stop, suspend, resume, migrate

High-QoS workloads must not suffer form being multiplex with low-QoS

Outline

Resource and Job Model

Scheduling problems are researched in the context of the following model

- Infrastructure provider P
- Hosts a set of m identical machines M_j ($j = 1, \cdots, m$)
- Machine are able to execute any job from the set of n jobs J_i ($i = 1, \cdots, n$)
- Machine processing capacity s_j is $\forall i, j \in \{1, \cdots, m\} : s_i = s_j = 1$.
- A job models the execution of a virtual machine instance
  - Load patterns vary over time
  - Jobs are sequential
  - Release time r_i and duration p_i

Resource and Job Model

We consider two types of QoS levels for jobs

- High-QoS jobs
  - Must start at time r_i
  - Should be able to allocate the full processing power of the machine
  - Are not preemptible
- Low-QoS jobs
  - Can be started at any time
  - Can be preempted at a fixed cost c_p.
  - A resumption of a virtual machine instigates a cost c_r.
- The job startup costs (c_b) and termination costs (c_t) are also modeled
- An example of a low-QoS workload is a VM that executes low-priority CPU intensive batch jobs.

Resource and Job Model

- Machines correspond to a virtualized core of a server
- Infrastructure provider *P* hosts a cluster of servers
- Machines can accommodate more than 1 job at a time
- Distribution and allocation of virtual cores to VM is handled by the VMM
- Simple initial model
  - Focus on CPU usage alone
  - Do not model multiplexing overheads
  - I/O contentions, cache line misses...

VM Management Model

Managing VM distribution over multiple servers requires a *virtual infrastructure manager* (VIM)

- The VIM determines the available features
- Many different options
  - vSphere, Eucalyptus, OpenNebula, ...
- We chose OpenNebula
  - Open source, research platform, feature set generality...
  - Haizea scheduler used a basis for simulation
  - VM operations: *shutdown, start, suspend, resume and migrate*

Simulation framework

- Using the Haizea simulation backend
  - Discrete event simulation
  - Supports job type differentiation
- Solve underutilization using an overbooking approach
- Scheduler does not know runtime
- Active scheduling manages low-high QoS interference
- Compatible with OpenNebula

Outline

Algorithm

- Reduce resource wastage while having a minimal impact on existing resource users.
- It is reasonable to assume high-QoS VMs do not continuously require all resources
- Goals
  - Only launch low-QoS when resources are available
  - Remove when interference might occur
  - Preserve performed work
    - VMs are uniquely suited (suspend, resume, migrate)
    - Overhead must be taking into account

Algorithm

- Simple and effective method to put restrictions on overbooking tolerance
  - Using just 3 parameters
  - Maximum amount of overbooked VMs
  - Lower bound: maximum server utilization when adding additional overbooked workloads
  - Upper bound: when should VMs start suspending
- Two steps:
  - Schedule new overbooking requests on suitable servers
  - Evaluate running requests and take appropriate actions if needed

Algorithm

Input: Set of nodes, Set of *vm_requests*, *lower_bound*
foreach *Node i* **do**
    if *Utilization(i)* $\leq$ *lower_bound* **then**
        available_nodes.add(*i*) ;
    end
end
Update(*vm_requests*) ;
while *available_nodes remaining & vm_requests remaining* **do**
    vm = *vm_requests*.pop() ;
    n = *available_nodes*.pop() ;
    Schedule(*vm* on Node *n*) ;
end

Algorithm 1: Adding Overbooked VMs

Algorithm

Input: Set of nodes, *upper_bound*

```
foreach Node i do
    if Utilization(i) ≥ upper_bound then
        vm = overbooked_vms(i).get_last() ;
        Suspend(vm) ;
    end
end
```

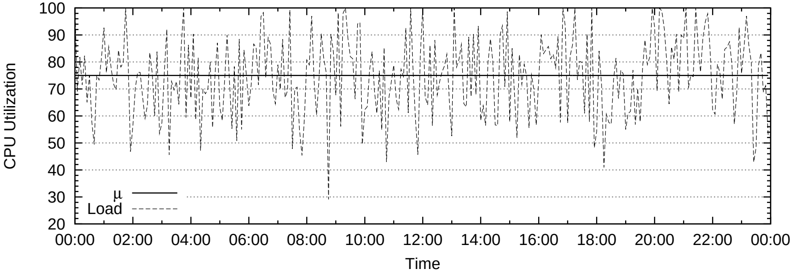
Algorithm 2: Suspending Overbooked VMs

Outline

Load Patters

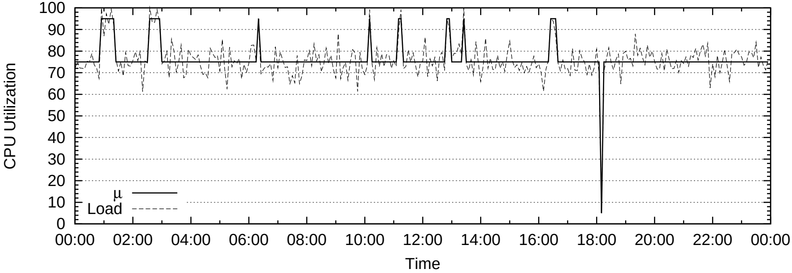
Three different load patters: noisy, spiky, business

Noisy: Starting from a mean utilization value μ, cpu load is generated by a normal distribution $N(\mu, 15)$.



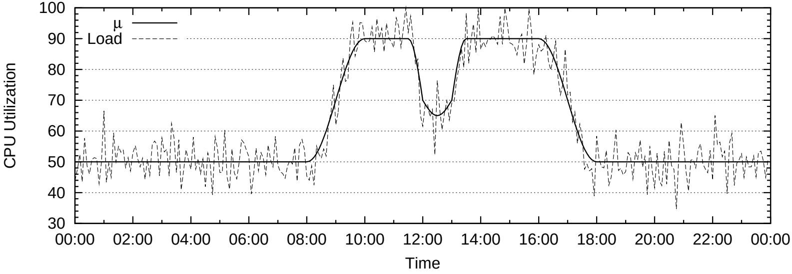
Load Patters

Spiky: This load pattern is based on a normal distribution with $\sigma = 5$. To add load spikes to the pattern, each drawing of the load distribution has 1% chance of generating a spike with 90% chance of having a positive one. Each spike has 50% chance of continuing.



Load Patters

Business: A function is used to determine the μ parameter of the normal distribution $N(\mu, 5)$ depending on the time of day. The value of μ is calculated with a piecewise function that represents utilization fluctuations coinciding with business hours.



Setup

- 50 homogenous octacore nodes
- Non-trivial synthetic load patters
- High-QoS
  - Each high-QoS application has an equal chance of generating one of the three load patterns
  - For spiky and noisy load patterns, μ is drawn from a normal distribution $N(75, 15)$.
  - The business load pattern, $min = 50$ and $max = 90$.
  - All cores are continuously occupied with high-QoS jobs
- Low-QoS
  - Each low-QoS job has a noisy load pattern with $\mu = 90$ simulating CPU intensive batch jobs

Setup

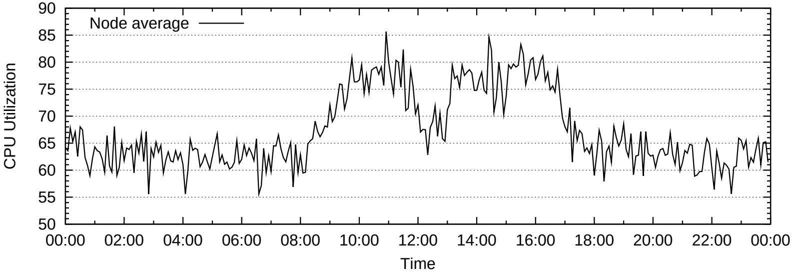


Figure: Sample load pattern on a single eight core node during a weekday.

Setup

All application runtimes are generated according to a geometrical distribution. If X is the runtime in minutes, the probability is expressed for $n = 30, 60, 90, ...$ with p equaling 0.1% and 1% for respectively high- and low-QoS applications.

$$Pr\left[X = n\right] = p\left(1 - p\right)^{\left(\frac{n}{30} - 1\right)} \qquad (1)$$

The costs for VM operations were configured as $c_b = c_p = c_r = c_t = 30s$.

Setup

- Executing without overbooking gives an average utilization of 69.4%
  = fairly high average utilization
- Every test is a variation on three parameters:
  - Max overbooked VMs either 1, 2 or 3
  - Upper bound in step of 5 between [85,95]
  - Lower bound in step of 5 between [60,80]
  - Minimum difference = 15%
  - Difference between bounds: window size

Outline

Results

Average utilization can increase from 69.4% without overbooking to:

- 73.7% with max 1 overbooked VM and bounds [60,85]
  $\sim$ 400 suspends = 1 per day/server
- 87.3% with max 3 overbooked VMs and bounds [80,95]
  $\sim$ 8800 suspends = 25 per day/server

Results

Some trends can be observed across the different bound selections:

- With low bounds max overbooked VMs has low impact
- Moving from max 1 to 2 overbooked VMs results in fewer suspend for similar utilization gains
- Increasing from max 2 to 3 results in more suspends an little to no utilization gains
- For the current setup we find that max 2 is the optimal choice

Results

Increasing the lower bound with a constant higher bound:

- Utilization gains decrease slowly
- Suspends increase exponentially
- Lower bound determines suspend/utilization factor

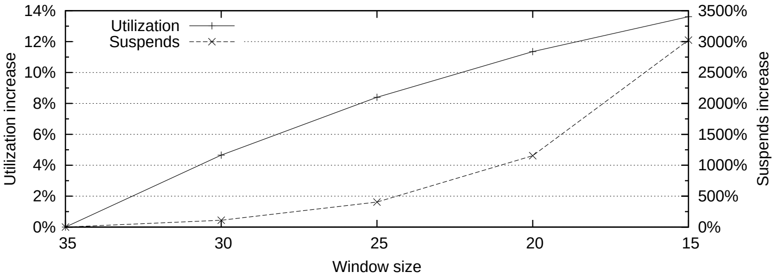


Figure: Increase in utilization and suspensions when using 2 overbooking slots and an upper bound of 95. The lower bound is increased to decrease the overbooking window.

Results

Increasing the upper bound with a fixed window size results in:

- A linear increase in utilization

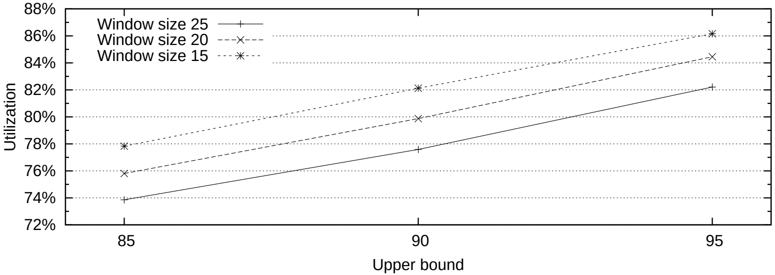


Figure: Utilization with two overbooking slots and varying upper bounds.

Results

- Roughly the same amount of suspends

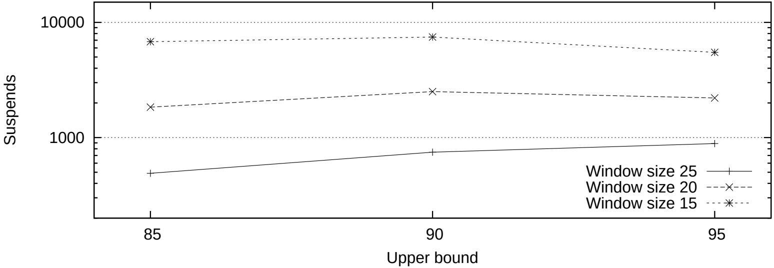


Figure: Suspensions with two overbooking slots and varying upper bounds and windows.

The selection of a correct upper bound will depend on factors not yet explored in the current simulation.

Outline

CPU intensive benchmark

- Dual socket quad core Intel Xeon
- CPU limited

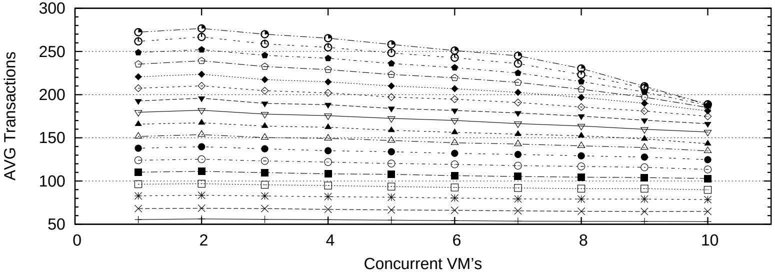


Figure: Sysbench scaling using both cpu and rate limiting with increasing VM amount

Analysis

- With 8 VMs we top off at about 240 transactions/second and 95% utilization per VM
- With 9 VMs we top off at about 220 transactions/second and 87% utilization per VM
- With 8 VMs, 220 transactions generates about 85% utilization per VM
- This is roughly the most conservative setup in the simulator which gained about 4% utilization

Outline

Future Work

- Add a more complex VM/VMM interaction model
- Use real world load trace data
- Create more accurate model (memory, network, ...)
- Implement complexer scheduling algorithms
- Compare with real world experiments using OpenNebula

Outline

Conclusion

- There are opportunities to increase utilization by monitoring the difference between formal and actual requirements
- Low-QoS workload overbooking can lower underutilization while having a manageable impact
- Scheduling policies can be simple and effective using a limited number of parameters
- An optimal selection of parameters can be made depending on the requirements

Questions?